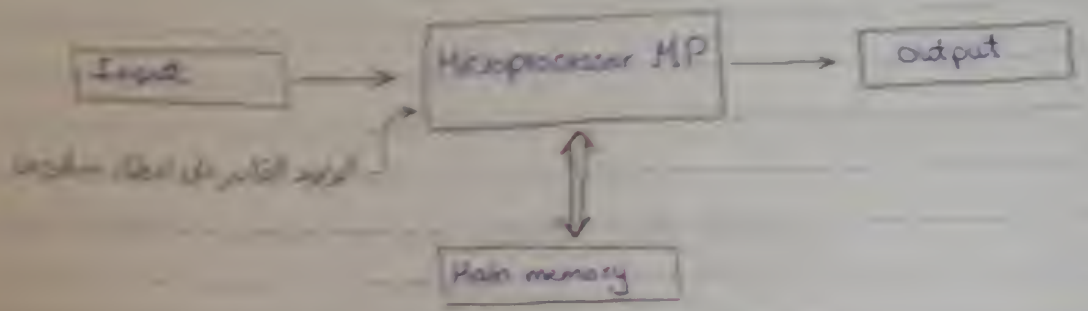
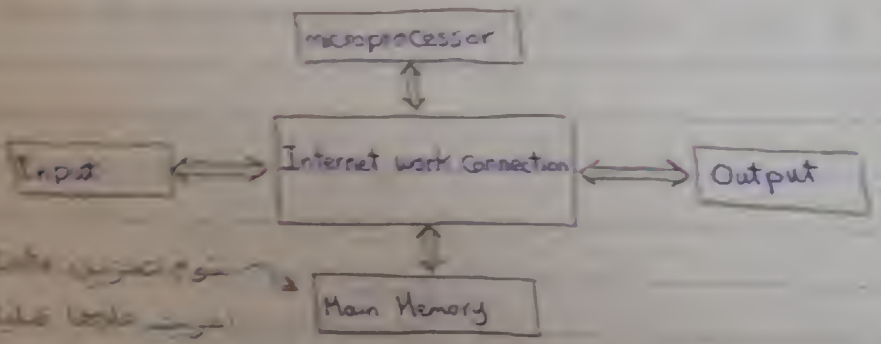
Ch 1 Microprocessor (Introduction)





Control processing Unit (CPU) ≡ Machine

→ CPU Components

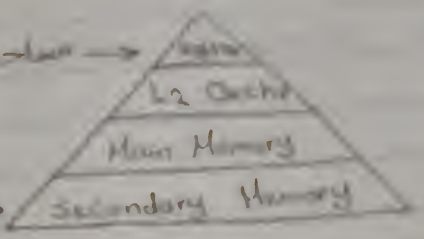
1. ALU
2. CU (Control Unit)
3. Registers level 1 Cache

تتحكم في وظائف الجهاز

→ CPU functions:

1 - Execode arithmatic & logic instruction
(Fetch - decode - execode)

→ Manage the program flow (make decrsions)
* Zero * negative * overflow ...



→ Transfer Data between itself and memory

Processor

Multi_processor ← → Multi_Core

Bus: group of common wires connect between parts of device.

1) Address Bus: →

K bit ⟶ 2^K devices

2) Data Bus: ↔

devices ↔ MP

كلما زاد كلما زاد معدل نقل البيانات من والى الـ MP

3) Control Bus: ↔

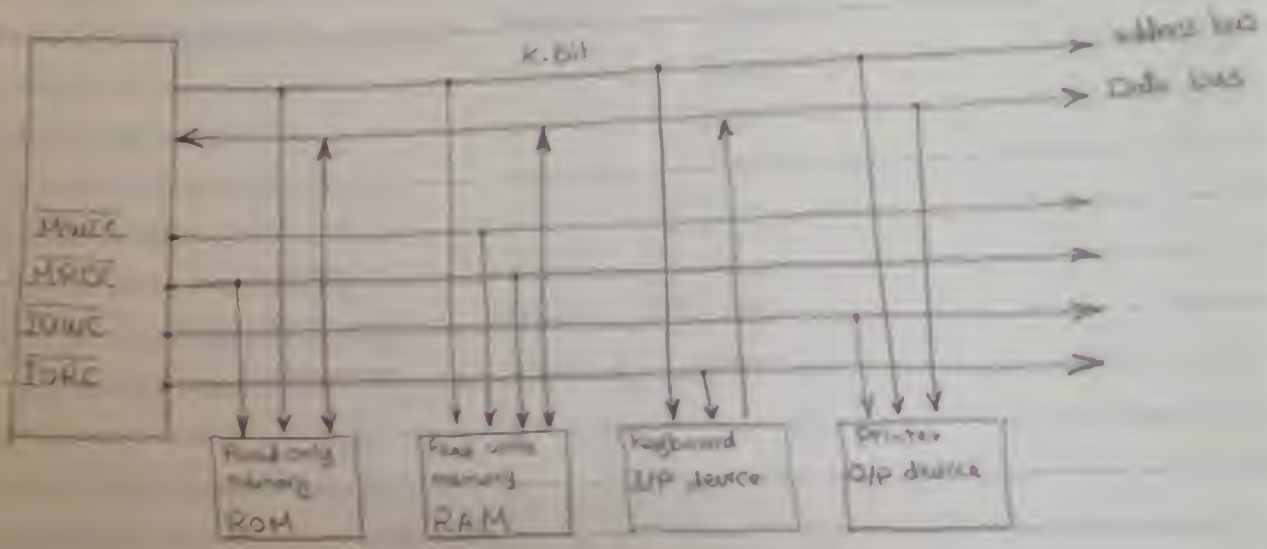
devices ↔ MP

Control Signal ينقل

كلما زاد كلما كان أفضل

←

Draw the block diagram of a computer system showing the address data and control bus signal ?



* Simple control Signal

MWIc memory write control signal
MRDC memory read control signal
IOWC I/O write control signal
IORC I/O read control signal

→ MP → device التفاعل بين

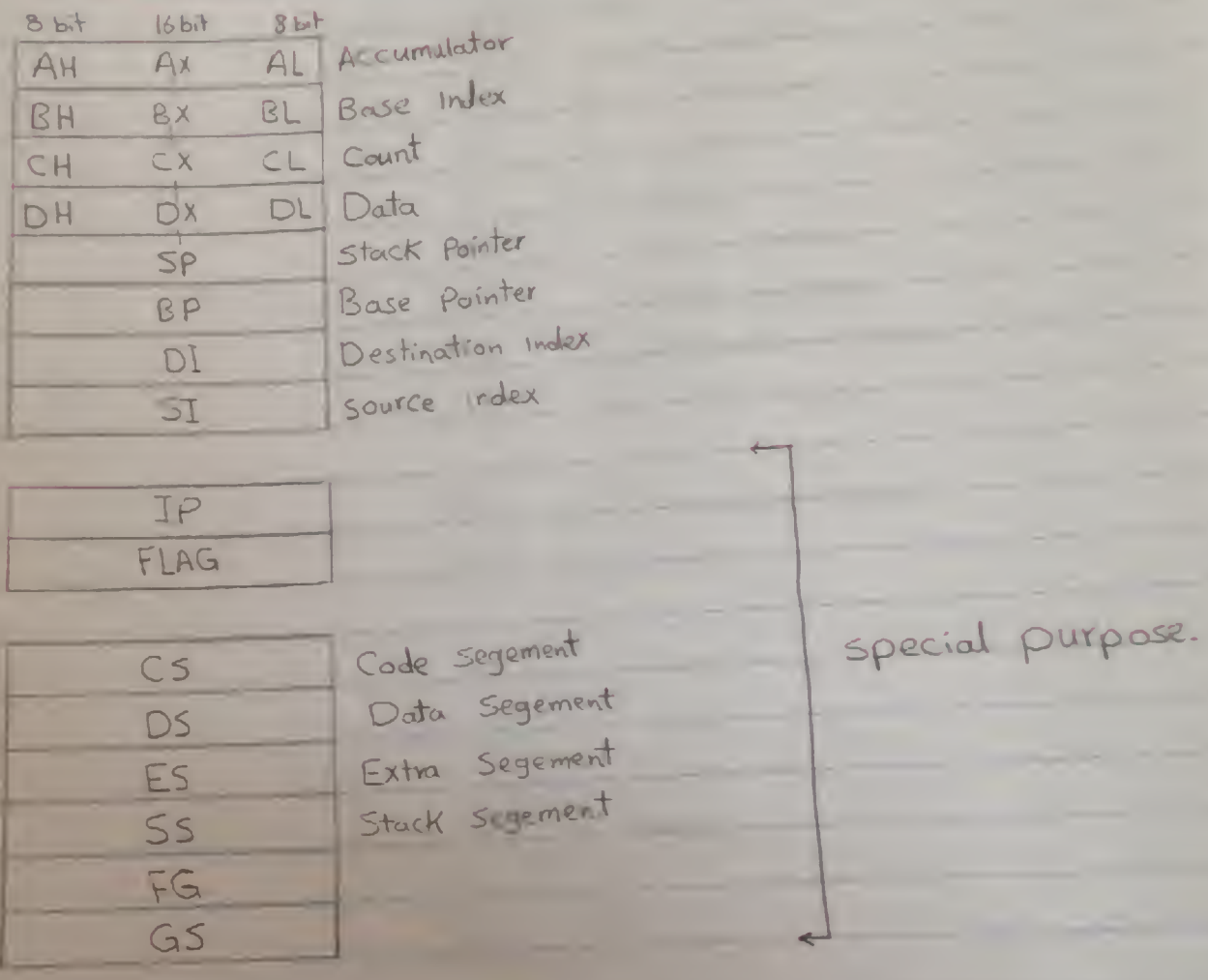
1- MP ⇝ Send address
2- Control Signal MRDC
3- Data transfer.

The Memory System is divided into 3 main parts:

1. Transient program area (TPA)
2. System Area
3. Extended memory System (EMS)

2.
- → holds
- → The length of TPA is 6400 KB
- → programs → interrupt vector

Ch 2 : The Microprocessor & Its Architecture



Programming Model Of The Intel 8086 Pentium 4

32 bit		16bit	8bit	
EAX		AH	AX	AL	Accumulator
EBX		BH	BX	BL	Base Index
ECX		CH	CX	CL	Count
EDX		DH	DX	DL	Data
ESP		SP			Stack Pointer
EBP		BP			Base Pointer
EDI		DI			Destination Index
ESI		SI			Source Index

	IP	Instruction Pointer
	FLAGS	Flags

CS	Code
DS	Data
ES	Extra
SS	Stack
FS	
GS	

The Programming Model OF The Intel 30386 Pentium 4

كل 4 bit في الشاشة بعطي 1 bit في السادس عشر ←

20 bit → (XXXXX)$_{Hexa.}$

Real Mode

Segment	Off Set	Special Purpose
CS	IP - EIP	Instruction address
SS	SP - BP ESP / EBP	Stack address
DS	BX - DI - SI 8 bit / 16 bit / 32 bit	Data address
ES	DI , EDI String Instructions	String destination address
FS - GS	No default	General address

Programming Model

program Visible
- They are used during app-prog
- EAX, EAX, AX, AL, EBX ...
→ Mov Ax, BX

program invisible
- not addressed directly during app-prog
- only 80286 - P4 Contain this type
- ex: FLAG, EFLAG

SUB Ax Bx بعد تنفيذ أمر

Purpose of use

General Purpose
EAX , EBX
Hold Data address or off set +
البعد عن البدايه

Special purpose
CS, DS ES, SS

int X = 3
↑ address ↑ data

Addressing modes (How the memory is accessed)

Addressing mode ← كثيره لكثره طرق تنظيم البيانات داخل الذاكره
وظيفتها الوصول إلى الذاكره

1- Real mode 1 MB only
2- Protected mode upto 4GB

« Real mode »

→ Accumulator EAX . AX . AL . AH
Can be used as 32bit → EAX , 16 → AX , 8bit → AH, AL

II Special purpose.
← طرق أساسى فى عمليات الضرب والقسمة

→ Div AX , BX : $\frac{AX}{BX}$

AX: يوضع به الجزء الصحيح
DX: (Data register) يوضع به باقى القسمه "الكسر"

→ Mux AX , BX
16 * 16 ⟶ 32 bit

→ MUX EAX , EBX
32 , 32

DX (high bytes) AX (low bytes)

EDX (high bytes) EAX (low bytes)

Base Index

Can be used as 32 bit → EBX , 16 bit → BX
8 bit → BH , BL

General purpose → hold offset For Data segement

Note : In Real Mode

We have 64 KB processor , 5 bit memory

→ To access the memory

← processor

$64K = 2^6 \cdot 2^{10} = 12^{16}$

$(16\ bit)_8 \rightarrow (4\ bit)_{Hex}$.

↝ From 4 bit processor how to reach 5 bit memory
we add zero From the left side to Base

→ To reach to any address in the memory we need Base and offset

Base is in Code Segement

$$\text{Current address}_{(real)} = (CS)_0 + \text{offset}$$

Ex: CS = 4000 Hex , IP = 2342

Solution

current address = $(CS)_0 + IP = 40000 + 2342 = 42342$ Hex

Count

To Count in any process with loop we need big register

loop CX , ECX

Repetition CX

shift CL we need small number of loops

Data

Can be used as 32 bit → EDX 16 bit → DX
8 bit → DH , DL

General purpose → Dlu , Hux العمليات والقسمة

Stack pointer

Can be used as 16 bit → SP
hold the off set of stack segement (SS) ← الأولوية له

Base pointer

Can be used as 16 bit → BP
used as off set for stack segement (SS)

Destination Index

Can be used as 16 bit → DI له الأولوية بعد BI
hold the off set of Data segement

Source Index

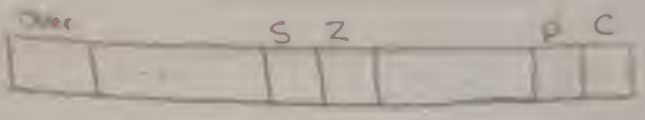
Can be used as 16 bit → SI له الأولوية بعد DI
hold the off set of Data segement

Instruction Pointer

Can be used as 16 bit → IP
hold the off set of current instruct to be excuted (in Code segement)

Ch 4 : Data Movement Instructions

C Flag - Invisible



Can be used as 16 bit - 32 bit

→ Help microprocesser to take the corect way

[1] Over flow happened when → +VE + +VE = -VE
→ -VE + -VE = +VE

[2] Carry (C)

Carry ① ← 11111010 + 11011000

[3] Parity (P)

1 → number of one's is even

0 → number of one's is odd

نستفيد من هذا عندما يتم نقل Data من مكان لآخر نتأكد من أن Data تم نقلها بدون فقد صح اذا كان عدد 1 زوجي بعد النقل وخالف قبل النقل تم بالصوره المطلوبه والعكس يكون به فقد في Data

Ex: SS = 3245 H
BP = 2342

Sol

Base address = 33450 H
Current address = 33450 + 2342 = 35792
Ending address = Starting + FFFF
= 33450 + FFFF

Used in Two dimension

Ch3 Addressing modes (Real Mode)

Examples

[1] Immediate Addressing Mode

I → R

Mov BL, 44 — Mov BH, 44H

Mov CL, 1001110 B

Mov ECX, FC23F34EH

[2] Register Addressing Mode

R ↔ R

Mov CL, BL ✓

Mov CX, BX ✓

Mov EAX, EBX ✓

Mov DS, CX ✗

Mov DS, CS ✗

Mov CS, BX ✗

Mov BX, CS ✓

[3] Direct Addressing Mode

R ↔ M (memory)

Direct addressing → R: EAX, AX, AH, AL

Displacement (off set) addressing → Any Register

Direct	Displacement
Mov AL, Num	Mov CL, Num
Mov AX, Num	Mov CX, Num
Mov Num, AX	Mov EBX, Num
Mov EAX, Num	Mov DX, [1023]

[4] Register Indirect Addressing Mode[]

└ (BP, BX, SI, DI)

R ↔ [M]

Mov AX, [BX]

Mov [BP], DL

Mov EDX, [DI]

Mov [BX], [SP] (X)

[5] Base_Plus_Index Addressable Mode

EBX → BX, EBP → BP

ESI → SI, EDI → DI

use with one dimention array

R ↔ M

Mov DX, [BX+DI]

Mov WORD / DWORD PTR [BX+DI], DX

[6] Register_Relative Addressable Mode

BP, BX, SI, DI

EBP, EBX, ESI, EDI

off set

R ↔ M

Mov AX, [BX+1000]

Mov AX, 1000[DI]

Mov WORD / DWORD PTR [BX+1000], AX

[7] Base_relative plus Index Addressable Mode

Used in Two dimention Array

R ↔ M

Mov WORD PTR 1000[BX+DI], A

Mov DH, [BX+DI+20H]

Effective Address (EA)

No EA

Notes

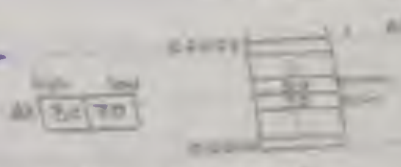
عشري ( )10
ثنائي ( )B
سادس عشر ( )H

R_{source} (R_S)
الـ Register اللي بحصل منه على البيانات

R ⟷ R has the same size
← لا يجوز بأي حال من الأحوال تغيير Code segment (CS) الذي يحتوي على عنوان الأوامر داخل الذاكرة وكذلك لا يجوز النقل من segement الى آخر

E.A = memory = Num

1→ Memory ⟷ Byte addressable
Byte by Byte عنونة

2→ 

i.We endian assignment

$EA = (seg)_{+0} + offset$
① $EA = (DS)_0 + BX$
② $EA = (SS)_0 + BP$ ③ $EA = (DS)_0 + DI$

← عند نقل بيانات من Memory الى Register لا بد من تحديد المساحة المطلوب تحويل البيانات لها حيث أن Memory byte addressable

لم يتم تحديد مساحة التحويل ← Mov [BX], AL
لم يتم التعديل الى ←

Mov Byte PTR [BX], AL
Mov WORD PTR [BX], AL
Mov DWORD PTR [BX], AL

ويظل effective address ثابت

$EA = (DS)_0 + BX + DI$
source address

$EA = (DS)_0 + BX + 1000$

$EA = (DS)_0 + \begin{matrix} BX \\ DI \\ 1000 \end{matrix}$

← عند نقل بيانات من Register الى Memory لا تحدد المساحة للتخزين فلا توجد مشاكل

Ch 4 : Data Movement Instructions

1- Assembler & Deassembler

2- Stack (push, pop)

3- Instructions

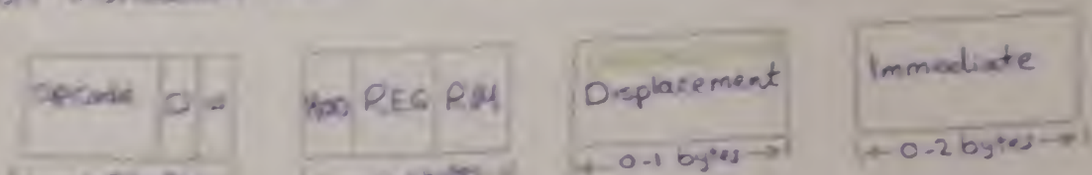
Assembler & Deassembler

Assembler: translates the mnemonic code to machine code (0's & 1's)

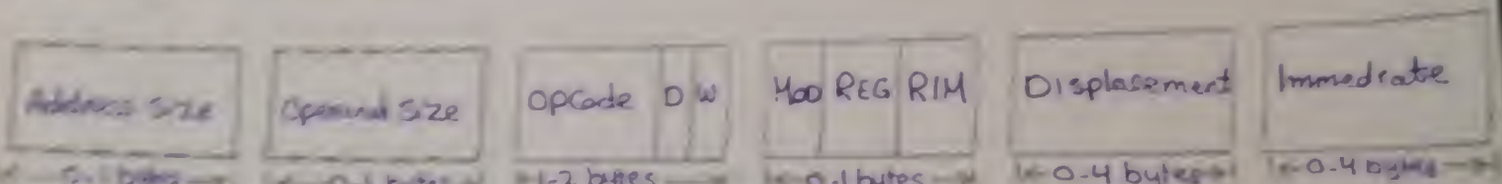
Deassembler: translates machine code to mnemonic code

Modes of Instruction

1. 16 bit instruction mode (80386)



2. 32 bit instruction mode (80386 above)



To change from default to desired follow the steps

1

Operand Size	Address Size	mode
66 H X	67 H X	- 16-bit instruction mode with its default (8 - 16 bit) registers
✓	X	- 16 bit instruction but works with (8 - 32 bit) registers.
X	✓	- 32 bit instruction works with its default (8 - 32 bit) register
✓	✓	- 32 bit instruction but works with (8-16 bit) registers.

2

Opcode	D	W

→ Opcode : has machine code of the instruction (Mov , Sto , Swap)
we get the code from the table
like Mov ——→ 100010

→ D ——→ D=1 (when we transfer from Memory to register)
——→ D=0 (when we transfer from register to Memory)

→ W ——→ W=1 (word / Dword)
——→ W=0 (Byte)

3

MOD	REG	R/M

→ MOD : its machine code can be get from this table ↷

MOD	Function	Ex:
00	No Displacement	Mov AL, [DI]
01	8-bit Sign-extended displacement	Mov AL, [DI+2]
10	16-bit displacement	Mov AL, [DI+1000h]
11	R/M is a register	

R/M is memory

→ REG : its machine code can be get from this table

Code	W=0 (Byte)	W=1 (word)	W=1 (Dword)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Code	Seg
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

→ R/M → to get the code of R/M we want first we want to know if R/M is Memory or Register not depend on MOD

IF MOD = 11 & R/M = registers
we can get its code from register table

Code	W=0 (Byte)	W=1 (Word)	W=1 (Dword)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

IF MOD = (00, 01, 10) & R/M is Memory
we can get its code
from this table

R/M code	Addressing mode	Function
000	DS:[BX+SI]	DS:[EAX]
001	DS:[BX+DI]	DS:[ECX]
010	SS:[BP+SI]	DS:[EDX]
011	SS:[BP+DI]	DS:[EBX]
100	DS:[SI]	Uses scaled-index byte
101	DS:[DI]	SS:[EBP]*
110	SS:[BP]* special addressing mode	DS:[ESI]
111	DS:[BP]	DS:[EDI]
	16-bit R/M memory addressing mode	32-bit addressing mode selected by R/M

[4] Displacement

① 1 byte [ ] → Put as it
ex: 1 → 01 21 → 21

② 2 byte [ ][ ] → Swap high : low

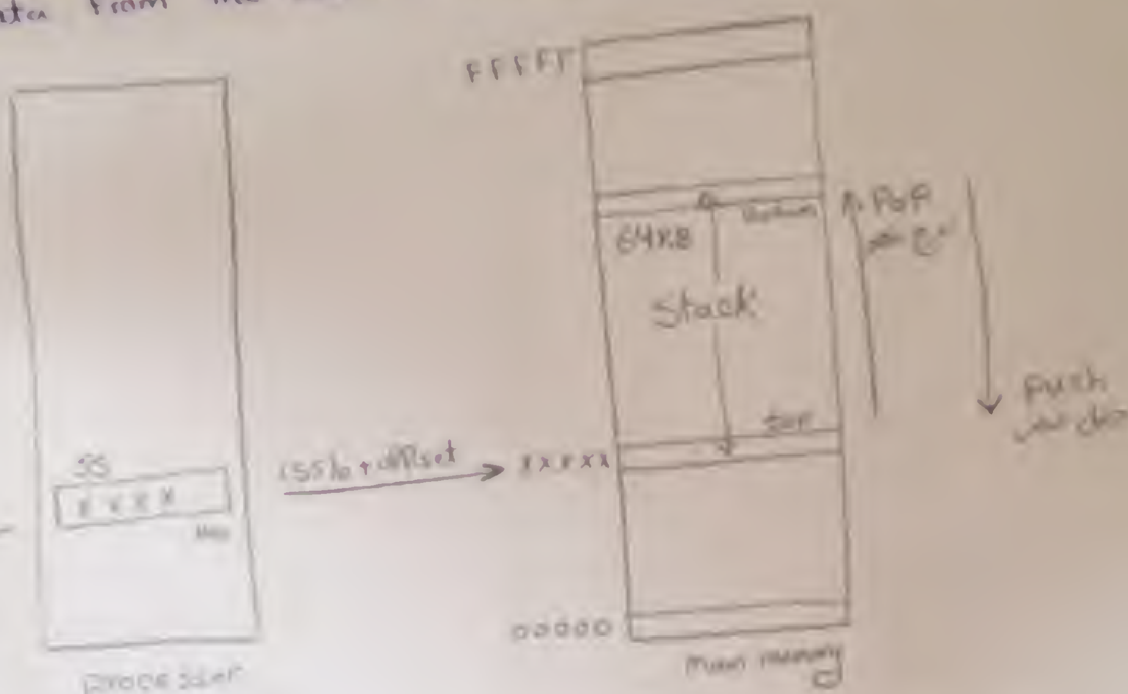
ex: 1234 → 3412

ex: 123 → 2301

Stack
(pop & push)

→ The PUSH and POP instructions are important instructions that Store and retrieve data from the LIFO (last In first Out) stack memory

Stack (LIFO)
↓
two operations
- Pop: يطلع عنصر ويزيد Pointer
- Push: يطرح Pointer ويدخل عنصر

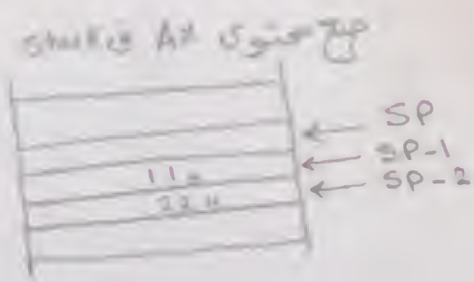


← عند ملء stack بالعناصر ويطلع عنصر آخر ما يحدث cyclic nature
ان يتم مسح ما بالعنصر
ويضيع العنصر القديم من مكانه

*

→ Push AX (✓)

AX | 11H | 22H | (high byte, low byte)



* Hint

22H can be Put in memory as?

1. Byte: 22H
2. word: 00 / 22
3. Dword: 00 / 00 / 22

*

→ push 5262H (✓)

→ push [BX] (X) ← هنا لم يحدد المساحة التي يخزن بها البيانات في memory

EA = (DS)0 + BX

the Correct

Push Byte PTR [BX] (✓)

Push (word - Dword) PTR [BX] (✓)

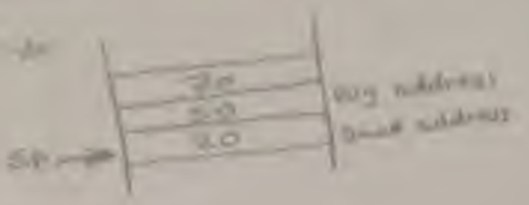
→ Push CS (✓) → هنا لم يغير محتوى CS

Pop ← stack or data

Pop 2345H (X)

Pop CS (X) ← we mustn't change CS

Pop BX (✓)



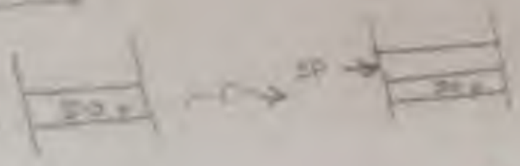
① BX | 50 | 20 |
high low

high address → high byte
low address → low byte

② Then SP = SP + 2

Pop Byte PTR [BX]

EA = (DS)₀ + BX



Instructions

- LEA
  - LDS, LSS, LFS, LGS, LES
  - * off set
- LODS, STOS, MOVS, INS, OUTS.
  - * IN, Out
  - * XCHG
  - * BSWAP
  - * CMOS

→ II LEA (load effective address)

Ex: LEA AX, Numb ≡ Mov Ax, off set Numb

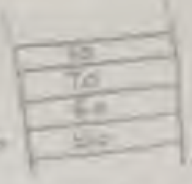
هذا الأمر معناه وضع بعد المكان Numb عن البداية في AX

ex: LEA BX, [DI] → EA = (DS)₀ + DI ≠ MOV BX, offset [DI] (X)

→ LDS

ex: LDS BX, [DI]

EA = (DS)L + DI = xxxx →



BX | 60 | 50 |

DS | 10 | 70 |

- يتم تحديد بداية العنوان من EA
- أخذ من memory بيانات تجميعها حسب المكان الذي سوف توضع له
في المثال BX ... 16 bit
4 byte = أربع خانات من الذاكرة
والتعبئة في BX حيث جميع العنوان
الأول من الذاكرة في BX وبعدها المكان التالي في ... byte
- ثم أخذ العنوانين التاليين والتعبئة في DS حيث أن الرمز LDS

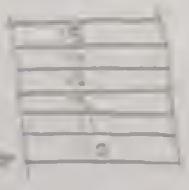
Hint

bit = 2^byte

→ LSS

EX: LSS EAX, [BX + DI]

EA = (DS)₀ + BX + DI = xxxxx →



EAX | 13 | 12 | 11 | 10 | EAX → 32 bit

SS | 15 | 14 |

→ LES

ex: LES BX, [DI]

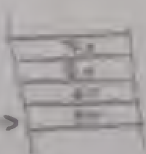
EA = (DS)₀ + DI = xxxx →

BX | 60 | 50 | , ES | 80 | 70 |

→ LFS

ex: LFS BX, [DI]

EA = (DS)₀ + DI = xxxxx →



BX | 60 | 80 | , FS | 70 | 50 |

→ LGS

ex: LGS BX, [DI]

EA = (DS)₀ + DI = xxxxx →

BX | 2 | 1 | , GS | 4 | 3 |

LCS EAX, [DI] (X)

because CS mustn't be changed.

String Instructions

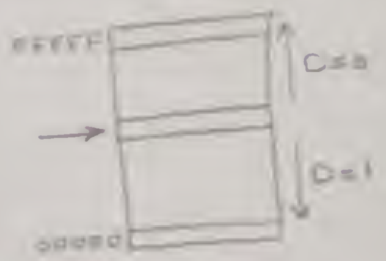
LODS, STOS, MOVS, INS, OUTS

This instructions depend on direction D

| | D | | FLAG

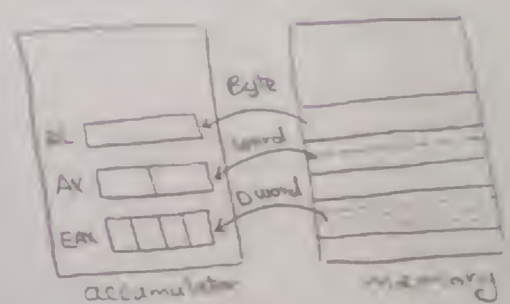
D=0 → autoincrement mode
تسير في اتجاه / بازدياد العنوان

D=1 → autodecrement mode
تسير في اتجاه نقص العنوان



1 LODS

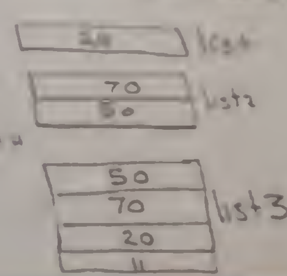
تأخذ من memory وتضع في accumulator
ثم أزيد offset او اقلله حسب D



LODSB → byte
LODSW → word
LODSD → double word
المكان غير معروف مباشرة بوصله عن طريق EA

LODS list → byte (DB)
LODS Data → word (DW)
LODS FROG → double word (DD)
المكان الذي تم وضع فيه في الذاكرة معروف مباشرة

list: DB 20H
list2: DW 5070H
list3: DD 50702011H



Source —to→ Destination

Data Segment — Extra Segment

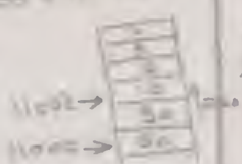
offset SI — DI

ex: DS = 1000, SI = 1000, D = 0
execute LODSW then LODSD

Solution

EA = (DS)$_0$ + SI = 10000 + 1000 = 11000

$SI_{new} = SI_{old} + 2$ bit = 1002



AX | 30 | 50 |

To execute LODSD

SI = 1002, DS = 1000, D = 0

EA = 10000 + 1002 = 11002

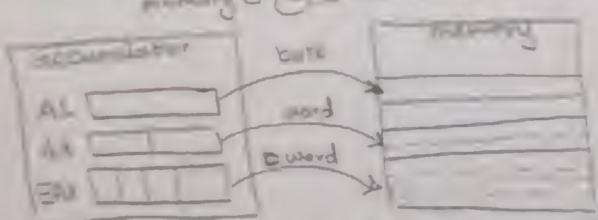
EAX | 7 | 6 | 5 | 0 |

$SI_{new} = SI_{old} + 4$ bit = 1002 + 4 = 1006

If D = 1 $SI_{new} = SI_{old} -$ 1 → byte, 2 → word, 4 → Dword

2 STOS

يأخذ من accumulator ويضع في memory



ثم يزود offset او يقلله حسب D

STOSB, STOSW, STOSD

STOS list (DB), STOS list2 (DW)

STOS list2 (DD)

3 MOVZX

يزود باقي خانات Register ب اصفار Zeros

MOVSX يزود باقي المساحة (Memory) بنفس القيمة حسب الاشارة

4 XCHG

بدل التبادل

→ Both places must have the same size

→ CS mustn't be Change

Examples:

XCHG AL, CL (✓)

XCHG CL, BP (✓)

XCHG Data, AX (✓)

XCHG DS, ES (✗)

XCHG CS, AX (✗)

5 BSWAP

→ Work only with 32 bit registers

80486 → P3, P4

EAX | 50 | 20 | 30 | 40 | before

EAX | 40 | 30 | 20 | 50 | after

6 CMOV (Conditional Move)

نقل مشروط

CMOVZ → هذا الأمر معناه
1- يرى حالة Flag ال Z
Z = 0 لا ينقل شيء
Z = 1 ينقل

ex: CMOVZ AX, BX

if Z = 1 ∴ AX = BX

else Do nothing

→ CMOVC → Chick Carry

if C = 1 → Move

else Do nothing

→ CMOVNC → Chick No Carry

if C = 0 → Move

else Do nothing